AFIT/GCS/ENG/90D-3



# A REMOTE VISUAL INTERFACE TOOL FOR SIMULATION CONTROL AND DISPLAY

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Computer Science)

William James DeRouchey, B.S.
Captain, USAF

December, 1990

AFIT/GCS/ENG/90D-3

# A REMOTE VISUAL INTERFACE TOOL FOR SIMULATION CONTROL AND DISPLAY

THESIS

William James DeRouchey
Captain, USAF

AFIT/GCS/ENG/90D-3



Approved for public release; distribution unlimited

## *Preface*

The purpose of this study was to research the current state-of-the-art in interactive simulation interfaces and to develop a generic graphical interface to support the Air Force Institute of Technology (AFIT).

I am endebted to many individuals for their assistance on this project. Specifically, I would like to thank the members of my committee for assisting me in the difficult times and reassuring me when I was doubtful of my ideas. I would like to thank the other graphics students for their assistance and guidance.

I would finally like to thank my wife Kathy and my 3 daughters Jennifer, Melissa, and Kristine, for understanding and supporting me through this gauntlett of horror.

William James DeRouchey

*Table of Contents*

## *List of Figures*

## *List of Tables*

## *Abstract*

This study discusses the design and development of the Visual Interactive Simulation Interface Tool (VISIT). VISIT was designed to provide the graphical representation and user interface for a simulation that is running on another processor utilizing internet communications.

The system provides support for various input devices to control the simulation display and environment. Simulation objects are displayed using either a three-dimensional wireframe representation or a Gouraud shaded representation. Viewer interaction to the simulation is provided by a collection of commands that allow the viewer to initialize, start, stop, abort, and restart the simulation. The viewer also has the ability to establish checkpoints. Upon reaching a checkpoint the viewer can step through the output display and/or manipulate the objects within the simulation.

# A REMOTE VISUAL INTERFACE TOOL FOR SIMULATION CONTROL AND DISPLAY

## *I. Introduction*

This thesis discusses the design and development of the Visual Interactive Simulation Interface Tool (VISIT). A visual interactive simulation as defined by O'Keefe is " a simulation which produces a dynamic display of the system model, and allows the user to interact with the running simulation " (16:461). VISIT provides the capability to graphically display the output from a simulation running on another processor. The simulation processor is accessible via network communications to the graphics processor. The viewer is allowed to manipulate and control the simulation through a collection of commands.

The methodology developed for this thesis is of particular interest because of its ability to generically interface to a wide range of simulations. It also provides the freedom of allowing the simulation processor to be any system capable of implementing the Transmission Control Protocol/ Internet Protocol (TCP/IP).

### *1.1 Background*

According to Biles a simulation is defined as "the development of a mathematical-logic model of a system and the experimental manipulation of the model on a digital computer" (3:7). A model is defined as "a representation of a system developed for the purpose of studying that system" (13) and Bratley defines a model as "a description of some system intended to predict what happens if certain actions are taken" (6).

For a model to be useful, it is imperative that all its relevant behavior and characteristics be determined in a feasible way. It may be determined numerically, analytically, or by running the model and providing to it input, and observing the results; the latter being a simulation. Although simulations provide a powerful tool for analyzing a given model, they are typically very large, application specific, and difficult to use.

Existing simulations are very large, having been built over time by numerous programmers. They typically initialize parameters, run for extended periods, and produce large amounts of paper output. There is either limited or no feedback presented to the viewer to identify how far the simulation has progressed. The simulations generally use limited graphics capability or none at all. They are developed for specific applications; any graphical representation incorporated into the simulation may require considerable redevelopment effort when the simulation is modified. Simulations that lack a visual display and interactive capability are difficult to use. If input parameters are entered incorrectly, the output will also be invalid, but considerable time and resources are wasted to generate that output. There is usually no interactive mechanism to abort a simulation gracefully if the viewer desires too.

The Air Force Institute of Technology (AFIT) needs an interactive visual interface to enhance and supplement its in-house simulation capabilities. AFIT has numerous computer systems that are capable of running large-scale simulations, but does not have a generic interface to display the running simulation graphically. Also, there is no capability to allow the user to interact with the running simulation. Clearly a standardized interface for use as a graphical representation tool for a given simulation would be beneficial.

## 1.2 *Thesis Statement*

A concurrently executing discrete-event simulation can be represented, controlled, and manipulated through a generic interface on a general-purpose graphics workstation, utilizing network communications with updates of at least 15 frames per second.

## 1.3 *Scope*

This thesis effort investigated the creation of a generic, three-dimensional, graphical interface to a simulation. The interface had to be generic enough to support multiple classes of simulations, but not so generic that a new graphics standard for simulations would be generated.

The simulation output is displayed in three-dimensional graphics and allows the viewer to specify any set of viewing parameters. The interface allows the viewer to specify a terrain or playing surface and icons to represent the simulation objects. All displayable objects (icons, terrain, etc.) use the AFIT polygon file format (see Table E.1).

The viewer can control the simulation by starting, restarting, continuing, and aborting it. To control the simulation display, the viewer can specify checkpoints, (specific instances during the course of the simulation where all relevant data is stored) as needed. Upon reaching a checkpoint, the viewer can step forward through the output display frames and query objects within the simulation. The viewer can also suspend the simulation to create, destroy, or reposition objects.

The display data is generated either from a simulation executing concurrently or read from an output file from a previous simulation execution. The data packets are transmitted using TCP/IP protocol over ethernet connections to the graphics display system (see Figure 1.1). The update packets are used

to extrapolate the position and orientation of an object from one position to the next. This information is extrapolated to graphically represent a smooth transition between the two locations.

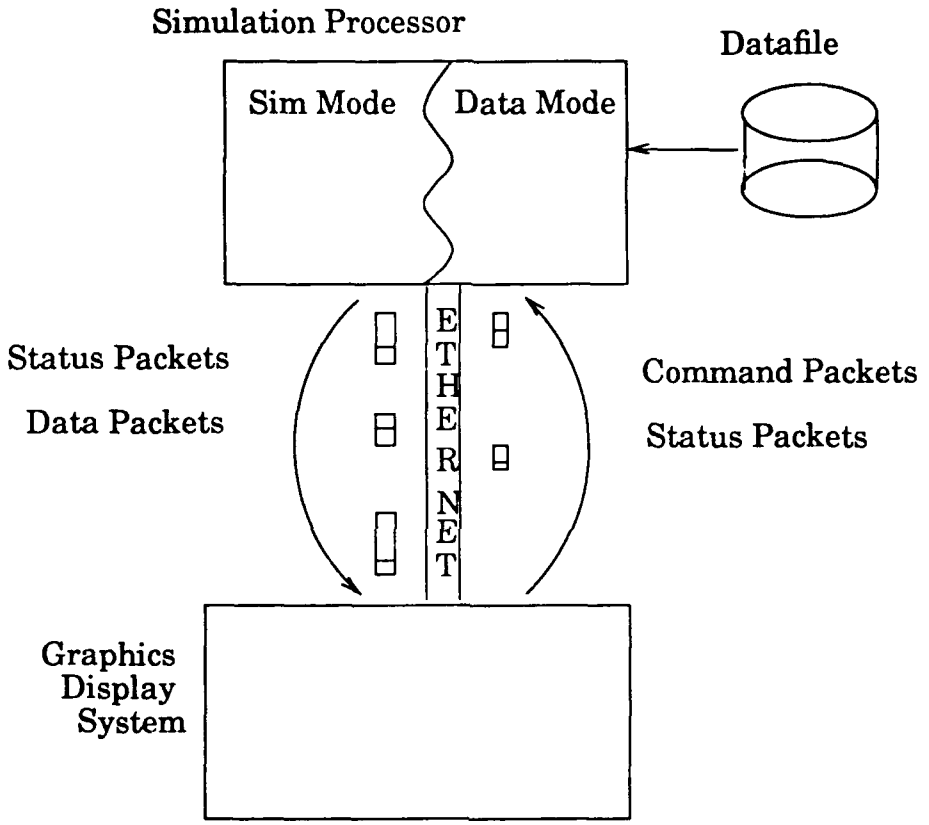


Figure 1.1. Network Message Traffic Flow

### *1.4 Assumptions*

Several assumptions were made in the analysis, design, and development of the Visual Interactive Simulation Interface Tool. The development of this software system could only occur if the supporting hardware and software were available.

As a minimum, the following equipment was assumed:

1. A graphics workstation with sufficient disk storage space, ethernet capability, and RS-232 communications port.
2. A TCP/IP protocol communications package for ethernet access.
3. A collection of 3-dimensional input devices, such as the CIS Dimension Six Spaceball and VPL Dataglove.
4. A system capable of executing the test simulation, with the software required to develop the simulation and access the ethernet via TCP/IP protocol.

## *1.5 General Approach*

The general approach for this thesis consisted of six major steps:

1. First, the literature search was conducted. From the literature search a familiarity with the categories of simulations was developed. Also, an understanding of the Visual Interactive Simulation approach was acquired. This approach provided the framework for the development of VISIT.
2. Second, the requirements analysis was performed for the software system. Since there was no sponsor other than AFIT, the requirements were identified by the author and validated by the thesis advisors. This task was extremely difficult in that there was no actual user to identify specific requirements.
3. After the initial requirements analysis was validated, the design of the interface began. An object-oriented design approach was utilized to benefit from the natural breakout of objects/entities within a simulation. This design incorporated a virtual environment software library (10) which allowed the interactive input/output processing to be delegated to a separate processor, thus freeing the graphics display from this task.

4. Next, the interface software was developed using C. The system was able to display the graphical representation on the graphics workstation while receiving the input from a simulation executing concurrently on another system accessible through an ethernet communication. An ethernet protocol package was modified to allow the simulation processor to be system independent, with the single restriction that it be capable of accessing a TCP/IP socket.

5. A test simulation was generated that provided the necessary conditions to test the interface. The test simulation had to support checkpoints, specific locations within the simulation identified by the viewer, where all state variables and events are saved. Additionally, simple test cases were developed for those capabilities not easily testable by a test simulation.

6. As time permitted, various external input device configurations were incorporated. These configurations allowed the external input device to communicate with the interface via two sources. One source was the graphics display system itself, the other was a separate Unix-based workstation which was connected to the display system through an ethernet connection.

### *1.6 Summary*

The execution of a concurrent simulation can be represented graphically, controlled interactively, and manipulated dynamically. The representation is accomplished using AFIT standard polygonal descriptions to represent the simulation objects in either a wireframe or Gouraud shaded display. The simulation is controlled by allowing the viewer to stop, start, restart, suspend, and abort the simulation. The manipulation is controlled dynamically through a standard, parameterized interface on a general purpose graphics workstation utilizing network communications.

The simulation interface functions much the same as an interactive symbolic debugger. The viewer is allowed to maneuver an object, remove an object, set checkpoints, and step through an execution. The checkpoints allow the viewer to "retreat" back to this position after following some path that proved undesirable. The idea of a checkpoint places a great burden on the simulation developer.

## *1.7 Thesis Overview*

The remaining chapters of this thesis represent the body of research developed in this effort. Chapter Two is a literature search that provides the information discovered while researching the specific topics of this effort. Chapter Three summarizes the requirements analysis and contains the design of the software system. Chapter Four discusses the actual implementation and assesses its utility and performance. Finally, Chapter Five presents the conclusions for this thesis and recommendations for future enhancements to the system.

# *II. Literature Review*

## *2.1 Introduction*

This chapter surveys current literature on topics related to this thesis. This review is limited to on-going research in computer graphics. Specifically, this paper briefly introduces the topics of Visual Interactive Simulation (VIS) and graphical interfaces.

## *2.2 Visual Interactive Simulation*

Simple animation to represent a discrete-event simulation has existed for many years. Amiry used animation in his simulation of a steel melting shop in 1965, but the present approach to VIS was developed by Hurrion (12) in 1976 at the University of Warwick, England. Hurrion was constructing simulations for job shop scheduling applications in manufacturing.

Hurrion determined that the human scheduler (the person responsible for scheduling the various machines in the manufacturing environment) held some control over the system, and the decisions made by the scheduler were sometimes difficult to portray in the simulation. Simulations were then created that gave the active scheduler control over decisions. This interaction required the scheduler to know the current system state. Letters were then added to the display to provide the system state of the entities to the scheduler. This approach allowed the scheduler to view the results of his decision.

Hurrion's initial applications represent model-prompted interaction in which the model prompts the user for a decision or additional data. Additional research at the University of Warwick created user-prompted interaction. User-prompted interaction allows the user to suspend the running simulation and thus control the interaction.

Other research and commercial development followed as the capabilities of VIS became known. Crookes (8) discussed the value of VIS for developers when they verify and validate a model. Interaction and animation provide instant feedback to the user and a mechanism to determine the validity of the model. By altering various parameters a user could observe the effects and compare them to what was expected.

Various packages have been developed that use VIS methods. VIS has dominated discrete-event simulation development in the United Kingdom during the past decade (2:109). The concept of VIS is not well known within the United States because VIS is centered around the user's ability to manipulate the running simulation. In the United States, animation, in this sense - the visual "playback" of a simulation, dominates visual simulation development (2:109).

A Visual Interactive Simulation provides the capabilities in a simulation for visual output, user interaction, and visual input. Although a VIS does not have to contain all of these elements, visual output and user interaction in some form are required, but visual input (the model is created visually instead of programmed or data-driven) is most often absent in present day VIS systems (16:461). Hurrion's initial work did not allow visual input.

There are three ways to incorporate visual output into a VIS package: embedded programming, automatic display, and animation. Embedded programming allows the developer to embed the statements for the visual output within the simulation. Automatic display is a technique in which the developer uses the generic display routines available from the operating system. Animation is the final visual output technique. Animation allows the developer to generate formatted output which is then decoded by an animation tool.

There are also three approaches to incorporate the user interaction

capability. They are embedded programming, standard interactions, and stopping interactions. Embedded programming is accomplished the same way as in visual output. Standard interactions make up a set of preconceived interactions that the developer may or may not include in the simulation. Finally, stopping interactions are provided where a VIS uses an interpreter for execution, thus allowing the developer to stop the execution at any line and modify the simulation.

The final capability of a Visual Interactive Simulation is visual input. This capability is provided by either graphical representations or iconic representations. The first method uses an intermediate graphical representation to describe the simulation model. The second method uses placement of icons and their connectivity to describe the simulation model (16:462).

Bell states that there are four major issues in VIS which require further research. These issues are the type and quantity of the virtual display, software and hardware for VIS, the need for methodology, and the role of expert systems (2:114). This effort will address the first two issues only. The importance of good screen layout is known to developers, but few developers have been trained in interactive computer graphics design. Present systems require the development of a simulation model first, then incorporate the graphical interface and interaction. VIS has typically been developed on stand-alone IBM-AT class computers with limited graphics capability (2:110).

Additionally, user interaction may compromise data integrity. A user can easily alter a system model by interacting with the model and emphasizing his or her own preconceived expectations. It is difficult to precisely "reproduce" a series of user interactions, thus causing the accumulated statistics to become invalid. However, a given simulation can constrain user interaction and thus preserve the integrity of the data.

## 2.3 *Graphical Interfaces*

The way a problem is presented influences one's understanding and ability to solve it. Researchers in human factors have emphasized the suggestive power of visual representations for a long time. Duisberg states "Solving a problem is simply a matter of representing it so that the solution is transparent " (9:305).

Animation is one way of providing a visual representation for the human-computer interface of a software system. A human's perceptual capabilities are optimized for real-time image processing. Interactive graphics can communicate instantly data in multidimensions concerning the internal state of a dynamic process. A fully interactive animation must interpret user input in relation to screen images, in addition to creating the images in the first place (9:300). The field of human-computer interaction combines many disciplines: computer graphics, human factors, cognitive psychology, and artificial intelligence.

Many mechanisms to construct specific graphical user interfaces have been developed. However, the most interesting classes of graphical user interfaces, namely those that involve concurrency, distribution, real-time control, or direct manipulation, remain difficult to construct (11:321). The advent of networks of loosely coupled workstations and of cheap, tightly coupled multiprocessors is beginning to change the nature of graphical computing. Applications are increasingly incorporating parallelism and often involve simultaneous multi-user interaction. There continues to be a need to deal with concurrent interaction with input/output devices.

An increasingly popular complementary technique involves the use of object-oriented systems to model the relationships between the objects on the screen being manipulated and their internal representation. This approach is particularly successful at the presentation level, namely at the level seen by

the user. There exists a direct correspondence between the icons and images the user sees, and an underlying representation.

The overall progress in interface construction has been significant; however current systems fail to provide mechanisms that facilitate dealing with the central problem of graphical interfaces of the near future. These interfaces involve concurrent, distributed, directly manipulated graphical objects. Interfaces requiring the use of direct-manipulation gesturing techniques also pose a severe challenge to today's user interface construction methodologies.

## *2.4 Summary*

VIS is the most important advance in discrete-event simulation since special simulation programming languages were introduced in the late 1950's (2:115). VIS development is becoming more popular in the United States as a means to visually portray discrete-event simulations instead of simply providing a visual playback capability. User interaction is an important addition to the visual feedback that is presented to the user.

Graphical interfaces continue to provide excellent capabilities to a simulation but further work is needed in many areas. Concurrency, distributed output, real-time control, and direct manipulation are among them. Finally, the creation of a visual display to explore the outcome of a simulation provides a developer the environment to stretch his or her creativity to its limits.

# *III. Requirements Analysis & Design*

## *3.1 Introduction*

With the development of numerous simulation systems at AFIT, there arose a requirement to develop a display system capable of graphically representing a simulation. Additionally, there was a requirement to allow the user to control the running simulation. Thus, the requirement was to provide an environment which allows the user to graphically view, manipulate, and control a time-dependent simulation, in real-time.

This chapter details the user requirements for this thesis effort. As this effort progressed, requirements were added and modified to ensure a generic interface, friendly user interaction, and realistic display features. The beginning of this chapter discusses the functional requirements, which is then followed by a discussion of the non-functional requirements. Next the design criteria and decisions made during the creation of the software system are discussed. Finally, a summary of the requirements analysis and design is presented.

## *3.2 Requirements*

The following requirements were established:

1. Functional Requirements

   (a) **Categories of Simulations Supported.** This interface will support both time-driven and event-driven simulations. It treats time-driven simulations as a special case of event-driven, where the movement of an object is extrapolated from a known position with respect to the time increment instead of the event.

(b) **Classes of Simulations Supported.** It will support war-gaming and mission planning military simulations. It will support other simulations that utilize moving objects on a static playing field.

(c) **Real-Time Display vs Playback.** It will support displaying data from either a concurrently executing simulation or a datafile from a previous execution. Viewer interaction is not applicable when reading from a datafile.

(d) **Design/Representation of the Terrain.** A 3-dimensional specification will be utilized by the graphics display system for the terrain/playing surface.

(e) **Design/Representation of Simulation Objects.** The system will support the ability to link each object within the simulation to an icon for the display. A list of icons and their graphical representations will be maintained on the graphics display system. The viewer may incorporate new icons into this list. All icons will be in accordance with the AFIT polygon geometry file format (see Table E.1).

(f) **Movement of Objects.** It will support a smooth movement of objects, based on position and orientation, that is proportional to simulation time.

(g) **Object Instantiation.** The interface will support simulations that have static and movable objects. Static objects are objects that never move or change, such as the terrain/playing surface. Movable objects are those that will change position during the simulation. All objects known by the graphics display system are viewable.

(h) **Viewing Control.** The viewer will be allowed to establish checkpoints during execution. Checkpoints are specific times during the course of the simulation at which all relevant data are stored. Upon reaching a checkpoint, the viewer will be allowed to manipulate

the simulation, return to a previous simulation checkpoint, step forward through the display, continue the simulation, or restart the simulation.

(i) **Viewer Interaction.** The viewer will be allowed to interactively suspend the display of the simulation. When the display is suspended, the viewer will be allowed to move objects within the display, delete objects, or select objects to use as a viewpoint reference. The viewer will be allowed to interact with any object known to the graphics display system and displayable from the current viewpoint. The viewpoint may be changed during the suspension.

2. Non-functional Requirements

(a) **Network Communications.** This interface will be developed to run on a Silicon Graphics Iris 4D workstation. It will support data communications that utilize the application layer TCP/IP protocol package. All communication between the simulation and the graphics display system will take place on the existing ethernet LAN in place within the graphics lab.

(b) **Frame Update Rate.** The graphical display will support rendering 25000 Gouraud-shaded polygons. The update rate will be at least 15 frames per sec.

(c) **User Interaction Devices.** The interface will support interaction devices other than a mouse and keyboard. The ability to pick an object in a 3-dimensional display will require a 3-dimensional picking device.

(d) **Coordinate System Supported.** The interface will support simulations that utilize the right-handed cartesian coordinate system to generate position and orientation data.

(e) **Time Scaling.** The viewer may specify a time scale factor to control the real-time display clock with respect to the actual simulation clock.

### *3.3 Design Overview*

The remainder of this chapter covers the rationale for the design decisions made during the development of this software interface. With the previous section requirements in mind, several general design considerations were developed. These general considerations are discussed first. Then communications aspects are covered, followed by user interaction considerations. Data representation issues and graphical concerns are the final topics discussed.

It should be stated that the breakout of the design considerations into the aforementioned categories is **not** disjoint. Data representation, graphical modeling, and display issues have interrelationships in the design of this system. Therefore, portions of the discussion of these topics will be interspersed. The ordering of their presentation has no bearing on their importance.

Finally, the design of this software interface involves multiple computer systems and their descriptions are presented here. The expression "graphics display system" refers to a computer workstation with a color graphics monitor and specialized graphics hardware. The term "simulation processor" refers to any general-purpose processor capable of executing a simulation and communicating over an ethernet network. The term "local area network (LAN)" refers to an ethernet based network on which both the graphics display system and the simulation processor can communicate. These topics are discussed further in the remainder of this chapter, but a general understanding is essential to continue further.

### 3.3.1 General Considerations

*3.3.1.1 Hardware.* The graphical display system used for the development of this software interface was a Silicon Graphics Iris 4D/85GT. This system utilizes the UNIX operating system and provides a software development environment using the C programming language. The C language was used for development to allow easy integration of this interface to existing network and input device communications packages. The 4D/85GT system provides a monitor capable of displaying 1280 by 1024 pixels of 24 bit color values (18). The system also utilizes special graphics hardware and a complete graphics library to provide the capability to generate and display 3-Dimensional geometric shapes at a rate sufficient to handle the requirements for this software interface.

Other graphics display systems were available. However it was shown in previous thesis work (22:15) that their performance and capabilities could not exceed those of the Silicon Graphics 3130. The SGI series 4D is a successor to the model 31xx series with improved capabilities and faster performance. The SGI 4D/85GT is rated at 90,000 gouraud-shaded independent, unlighted, 10 x 10 quadrangles per second (18).

The simulation processor used for the development of this effort was a Sun4/260 workstation. The performance and capabilities of this system are sufficient to meet the requirements of the simulation processor and it was already connected to the existing LAN. The LAN also contains a connection to the graphics display system.

*3.3.1.2 Generic Interface.* The interface between the simulation processor and the graphics display system had to be generic enough to support a wide range of simulations. The interface had to support the display of existing simulations without forcing a major rewrite of the simulation

software. To support this requirement, two modes of operation were designed, *datamode* and *simmode*. Datamode allows the graphical display system to display the results of an existing simulation by executing a software tool on the simulation processor. This software tool reads a datafile that contains converted simulation output. The contents of the datafile are sent to the graphical display system over the LAN using a prescribed message packet format. In this mode the simulation does not have to be altered if the required information for the datafile is available in the current simulation output. The simulation output must be converted into the datafile format externally (see Appendix C).

In the second mode of operation, simmode, the simulation is executing concurrently on the simulation processor as the graphics display system displays the output. The output, using the network message packet format, is provided to the graphics display system via the LAN that interconnects the two systems. This mode of operation provides more capability to the viewer but has the restriction that the graphical display system can only display the information at the rate at which it receives packets from the simulation processor.

*3.3.1.3 Coordinate Systems Used.* This software interface adopts the standard convention of using the right-handed cartesian coordinate system for the world coordinate system and the left-handed cartesian coordinate system for the eye coordinate viewing system. In the left-handed system the positive z axis points forward from the viewer's position, the positive y axis points upward, and the positive x axis is to the right (15:340). Since the simulation objects are described in one coordinate system and displayed using another, a transformation is applied to the geometric description of the objects before they are displayed onto the screen.

### 3.3.2 *Communications*

*3.3.2.1 Simulation Processor.* In order to support a wide range of simulations the graphics display system must communicate with a wide range of computer systems. The communications protocol has to be generic, reliable, and available. The Transmission Control Protocol / Internet Protocol (TCP/IP) satisfies these requirements. TCP/IP is a common term referring to the Defense Advanced Research Project Agency Internet network protocols developed for use on the ARPANET. TCP and IP are only two of the many protocols that were developed. The internet protocols support heterogeneous host systems and architectures that utilize a wide variety of internal data structures (5).

The Internet Protocol is the lowest-level protocol and provides the network-level services of host-addressing, routing, and packet disassemble and reassemble. All other protocols use the services of the IP. TCP provides reliable and controlled transmission of data from the application to the IP (14:344). A more detailed discussion can be found in (14).

TCP/IP provides the generic network communications between the simulation processor and the graphics display system in a reliable manner. The TCP/IP protocol is supplied and available on both computer systems used in the development of this software interface.

*3.3.2.2 External I/O Device.* To maintain a sufficient display rate on the graphics display system, user input and output had to function at a rate that would minimize the delay encountered using external I/O devices. The keyboard and mouse are standard input devices that provide efficient input and are already a fundamental part of the graphics display system. However, it is very difficult for a viewer to select an object from a three-dimensional display using a two-dimensional device. In order to provide this capability, an

external three-dimensional device is required. This external device requires cpu processing time for the system to read from the device and write to the device. An existing Virtual Environment Display System (VEDS) was available that provided the ability to offload this requirement to a separate processor (10).

The Virtual Environment Display System (10) is a system of software libraries and various hardware devices that support virtual environments such as VISIT. VEDS provides a means of allowing a separate computer system to handle the interaction with certain external I/O devices. The VEDS processor establishes a communication link to the device, continually reads data from the device, and stores the latest data from the device into memory. This memory is shared with another process running on the VEDS processor, which in turn establishes a network connection to the graphics display system.

In this environment the VEDS processor always has the latest data read from the device stored in memory. When the graphics display system requests data from the device, it is received over the ethernet connection from a memory location on the VEDS system, instead of polling an RS-232 communications line to the physical device. This ensures minimal waiting for data.

*3.3.2.3 Datamode Driver.* For existing simulations to be displayed on the graphics display system, a software tool had to be developed that would reside on the simulation processor. This software tool would read a datafile that contained the converted simulation output. The data would then be formatted into the records required by the graphics display system. A network message packet had to be developed that would contain the simulation data in a format known to both systems. The message packet header contains simulation-specific information to identify the quantity and type of information contained in the body of the packet.

*3.3.3 User Interaction.* A parameter file was developed to provide the user the flexibility to provide initial viewing parameters. These parameters allow the viewer to customize the environment for viewing a simulation.

*3.3.3.1 Keyboard.* The keyboard was utilized to allow the user the ability to modify the initial viewing parameters. This provides an environment for the user to experiment with different settings for various parameters and visually see the effects. The user may not have the computer graphics background and providing a flexible environment to modify and customize will allow the viewer to become comfortable and more willing to actually use the system. Since a mouse is becoming a more popular interaction device, popup menus are incorporated to provide the same functions as the keyboard.

*3.3.3.2 3-D Device.* In order to interact with a three-dimensional display the viewer needs a three-dimensional picking device. Selecting objects within the current display will require the viewer to manipulate a cursor over the object desired. Once positioned, pressing a button or key indicates to the interface select the object nearest the cursor. When objects appear to the viewer very close to each other selecting among them with cursor keys or a mouse becomes very difficult. In order to provide a picking capability, the system has to incorporate a three-dimensional device.

The CIS Dimension Six spaceball was chosen for its unique capabilities. The spaceball provides both the direction of movement and the amount of movement. This provides a fast and controlled movement of the cursor on the display screen along any of the three primary axes in either a positive or negative direction. In addition, the device provides the capability to provide three degrees of rotation in either a positive or negative direction.

*3.3.4 Data Representations*

*3.3.4.1 Simulation Objects.* The graphical display system and the simulation processor must have a common reference to the various objects within the simulation. To ensure that both systems uniquely identify all objects correctly, initialization records will be sent by the simulation processor to identify each object. The initialization links each object with a geometric description file for that object. The file contains the polygonal description which is used by the graphical display system (see Appendix E).

*3.3.4.2 Network Message Packets.* The graphical display system and the simulation processor will require an enormous amount of internetwork traffic. The amount of traffic is dependent upon the size of the simulation being displayed. A very large simulation containing hundreds, perhaps thousands of objects, will communicate one message packet for each update to each object. These packets are read by the graphics display system one at a time within the display loop. Small simulations, on the other hand, may communicate fewer message packets, and thus create less network traffic.

A message packet format was adapted from an existing internet communications package (see Appendix C). The message packet header contains a field which identifies the type of information contained in the packet body. Certain record types contain zero bytes of data in the body while others contain over 500 bytes of data. Since the interface reads the message header first to determine the record type and length of the message body, time is saved by reading the exact number of bytes in the body instead of having a default size large enough to contain any message body.

*3.3.4.3 Terrain.* Realistic terrain features were desired in the display of certain simulations. Flat terrain, although easy to generate and display, provides little realistic qualities to the displayed image. A three-dimensional description was desired in a format that was fairly simple to

incorporate into the graphics display system as well as the simulation processor. The terrain description was generated using the same geometric description format as the simulation objects (see Appendix E).

### *3.3.5 Graphical Display*

*3.3.5.1 Hidden Surface Removal.* There are many techniques available to handle the hidden surface problem in displaying computer graphics. Algorithms exist that provide a solution to hidden surface removal, but the hardware implemented Z-buffer available on the 4D/85GT was used for this thesis effort. The Z-buffer was chosen because it requires less software development, it is incorporated into the overall design of the graphics hardware, and it provides a fast method of hidden surface removal. The graphics display system contains an array of memory 24 bits in depth that is used to calculate the z value for geometric shapes within the current display. The Z-buffer maintains one location for each pixel on the screen. The buffer is initialized to the maximum value allowed and the color framebuffer is initialized to the background color.

During the rendering process, polygons, lines, points, and characters are converted to pixels that have an x, y,and z screen coordinate and a color. When the Z-buffer hardware is enabled, the z coordinate determines the distance from the pixel location to the eye position. The z coordinate is then compared to what currently exists for that pixel location in the Z-buffer. If the new z value is less than the value in the Z-buffer, the new z value is written into the Z-buffer and the new color value associated with that pixel is written into the framebuffer. If the new z value is greater than the value in the Z-buffer, it is discarded and the Z-buffer and framebuffer are not changed. Therefore, during the rendering process the Z-buffer maintains the distances to those items which are closest to the viewer and thus surfaces that are hidden by

others are eliminated (15:369).

*3.3.5.2 Object Movement Technique.* Event-driven simulations update the position of their objects at certain events which occur at non-uniform times throughout the execution of the simulation. As each object's location and/or orientation is updated, a network message is sent to the graphics display system. One way to display the objects would be to reposition and reorient the object whenever a new update is received from the simulation processor. This would be very efficient, but not very visually appealing. To provide a more realistic display of the simulation, the position and orientation of the objects must be interpolated between two successive updates. In this manner the position and orientation is updated a proportional amount during each update of the graphics display screen. The update represents the difference in the coordinates at the two positions, divided by the number of frames required to represent the objects in the simulation time needed to get from position one to position two. The orientation update is calculated in a similar manner.

Interpolation provides a realistic display when two locations of a given object are known. However, in an event-driven simulation, certain objects contain an initial position and orientation and may not be involved in any event throughout the simulation. These objects provide only one location, so interpolation cannot be used. Instead, a method of extrapolating the next position from a known position and velocity is used.

Extrapolation requires additional information be provided in the update records received from the simulation processor. Besides the information required for interpolation (time stamp, position coordinates, and orientation angles), the update records must contain the velocity vector components of the object's position and orientation (see Appendix C).

With this approach it is possible for object positions to be "overshot". If the next update of an object does not occur at an exact time increment used by the display system, a slight jump may appear on the display. Another, more severe case, occurs when the display system does not receive a location update record in time. If an update arrives and the time to activate that update has come and gone, the next display cycle will display the object "jumping" from its current displayed position to where it should be based on this late-arriving update record.

### *3.4 Summary*

The network communications was a crucial aspect in this software system. The structure of the message packets was required for the simulation processor to communicate with the graphics display system. After finalizing the animation technique, data requirements for the simulation processor were established. Various geometric descriptions were generated to model known objects for a given simulation. With the network message structure established, geometric description format finalized, and animation technique chosen; the software design was ready for prototyping.

# *IV. System Implementation*

## *4.1 Overview*

The overall strategy used in developing the implementation of the VISIT was one of prototyping. The project was broken down into three main areas during the design phase; data representation/processing, communications, and graphical display/user interaction issues. To handle this project in an orderly manner, a prototype for each major area was implemented and tested before they were integrated into a complete system.

The data structures representing the simulation objects, their positions, and orientations were considered first for development. However, the need to provide a network communications capability between the graphics display system and the simulation processor to transfer this data seemed the more difficult task and was undertaken first. The display issues, as well as user interaction concerns, were implemented last to ensure a working product would be established and, as time permitted, enhanced.

## *4.2 Communications*

To assist in the discussion of the implementation of the networking features, an overview of the hardware environment is presented here. The Silicon Graphics Iris 4D/85GT workstation is connected to a subnet on the LAN that is available throughout the AFIT complex. This subnet utilizes thickwire ethernet connections and provides a network file system (NFS) to all hosts. The simulation processor used throughout the development was a Sun 4 workstation, also connected to this subnet. A layout of the hardware architecture is shown in Appendix A.

Using an NFS environment, all development files were available to the graphics display system and the simulation processor. This provided a very efficient working environment since a window could be maintained on the graphics display system that was remotely connected to the simulation processor. Since both systems had access to the same file system, no files had to be repeatedly transferred between the two systems. This enabled virtually all work to be accomplished on one system instead of going back and forth between the two systems.

*4.2.1 Simulation Processor.* The simulation processor used in the development was chosen for its availability and because it was already connected to the same subnet. However, there was a requirement that the software system support a wide variety of platforms for the simulation processor. In order to provide this capability, an existing network communications package was used as a starting point. This package provided a client-server based relationship between two heterogeneous computer systems and was available for a Sun 4 and a Silicon Graphics Iris 3130.

The message packet format was modified to provide the necessary information in the header of the packet (see Table 4.1). This provided the capability for using variable size record formats. A single field within the message header holds the number of bytes contained in the message body. When messages are passed between the two systems, m bytes are read from the network socket first, then n additional bytes are read. The size in bytes of the message header is represented by m, while n represents the length in bytes of the message body. This allows for efficient network traffic by only sending the number of bytes required for this particular type record.

The flow of messages is bidirectional as shown in Figure 1.1. The simulation processor sends data records and status records to the graphics

Table 4.1. Message Packet Structure

| *Field* | *Description* |
|---|---|
| request | request type |
| reply | reply from cube |
| channel | values used in cube calls |
| node | reserved for future use |
| pid | reserved for future use |
| type | reserved for future use |
| maxlth | reserved for future use |
| cnode | reserved for future use |
| cpid | reserved for future use |
| ccode | reserved for future use |
| recvdlth | reserved for future use |
| object | object number |
| icon | icon index number |
| buflth | length of following buffer |
| buf[MAXBUF] | message data |

display system and the display system sends command records and status records to the simulation processor (see Appendix D). In order to maintain efficient processing, the graphics display system performs "nonwait" reads from the network socket. The system queries the network socket to see if any messages are pending; if none are available it continues on instead of waiting.

*4.2.2 Modes of Operation.* To provide a display system for existing simulations, two modes of operation were designed. In datamode the existing

simulation output, previously converted to a datafile in the format required, is read by a software tool residing on the simulation processor. In simmode the simulation creates message packets during the execution and passes them to the graphics display system for delayed realtime processing.

*4.2.2.1 Datamode.* The software tool was developed to execute on a Sun 4 workstation and is called Simtoge (SIMulation TO Graphics Engine). Simtoge is not Sun specific and will compile and execute on other Unix based systems. The tool functions as a preprocessor. It reads the datafile, creates the network message, sends the message, and waits for commands. The basic algorithm is as follows:

1. Initialize the network socket
2. Establish a handshake with the graphics display system
3. Open the datafile
4. While not end of file:
    (a) Check for incoming network messages
    (b) Read a record from the file
    (c) Create the network message packet
    (d) Send the message packet

Once the datafile is completely sent across the network, the system continually checks the network socket for incoming messages from the graphics display system. This provides the features and "hooks" needed by a simulation to communicate with the graphics display system. The tool is designed to be called from the graphics display system as part of its initialization procedure if required.

*4.2.2.2 Simmode.* In simmode the simulation processor will create the network message packets as it executes the simulation. The processor will have to perform some initialization tasks before beginning the execution of the simulation.

The basic algorithm is as follows:

1. Initialize the network socket
2. Establish a handshake with the graphics display system
3. Begin the simulation
4. Until the simulation ends:
   (a) Check for incoming network messages
   (b) Process an event
   (c) Create the network message packet
   (d) Send the message packet

## *4.3 Data Representations*

*4.3.1 Object.* The task of maintaining the location, orientation, and characteristics of all objects within the simulation requires a detailed data structure that can be easily manipulated and maintained. The data structure for an object is shown in Table 4.2. The various characteristics of an object (color, scale, min, max, centroid, plist, etc.) are established when the object is linked to an icon index. As update records are received from the simulation processor the array of locations is updated. During each frame update the current position of each object is updated and stored in the position field of the structure.

Table 4.2. Object Data Structure

| *Field* | *Description* |
|---|---|
| color | wireframe color |
| visible | flag to indicate object's visibility |
| scale | world coordinate scaling parameters |
| rotation | world coordinate rotation parameters |
| min | minimum x, y, & z coordinate |
| max | maximum x, y, & z coordinate |
| position | object's current world coordinates |
| distance | distance to the cursor in picking mode |
| centroid | object's center coordinates |
| type | display characteristic |
| display | flag to indicate if object is displayed |
| pcount | number of polygons in geometric description |
| plist | pointer to the list of polygons |
| objectId | object's identification number |
| iconId | icon number associated with this object |
| termination_time | simulation time when this object goes away |
| current_location | the current location record used |
| final_location | the last location record |
| locations | array of location records |

*4.3.2 Message Packet.* The message packet is the medium used to transfer data between the simulation processor and the graphics display system. The format of the message header is shown in Table 4.1. This format contains fields that were established for parallel processing applications and are not relevant to this thesis. These fields were maintained to allow this software system to be expanded to applications involving parallel processing simulations.

Several of the fields are used in almost every record passed from the simulation processor to the graphics display system. The list of record types and their format is located in Appendix C. The ordering of messages received from the simulation processor is critical to the graphics display system. The icon description record used to link the simulation object type to a geometric description file must be received first. A list of geometric description files available on the Iris 4D/85GT is contained in Appendix E.

Once all objects types that will be used by the simulation are known to the graphics display system, the object records can begin. When a new object is created in the simulation, an icon assignment record must be received. This record links the simulation object to an icon file that will be used to represent the object on the display screen. After the object is linked to an icon, the initial position record must be received. This record identifies the location, orientation, velocity, and simulation time at which the object will be at that location.

The graphics display system will wait to begin the display of the simulation until a special message packet (record type 50), used to instruct the display system to begin the display process, is received. The simulation should send this message after all known objects that have a location at time 0 are sent to the graphics display system. Once the display system receives this record it will begin to display and update all objects known to the graphics

display system. The method used to update the position of objects is discussed in Section 4.3.2.1. New objects can be created after the start of the display process.

The graphics display system sends network packets to the simulation processor when the viewer stops the simulation to manipulate a known object. When the viewer relocates or removes an object from the display system, this information is passed to the simulation processor. The simulation must then respond to those records and reply with object update records. In datamode these records have no effect on Simtoge. Simtoge, the simulation postprocessor, reads the message and displays an acknowledgment in the console window, but does not manipulate the object. The graphics display system will continue updating an object's position and orientation after it is relocated.

*4.3.2.1 Extrapolation Technique.* VISIT uses an extrapolation method to update an object's position and orientation during the display process. Given a known position, orientation, velocity, and time, an object's future position can be determined. This assumes a straight line movement and/or a constant rotation. Table 4.3 shows the format of the location structure used to extrapolate an object's position and orientation. During each frame update the system performs the following steps to update each object.

1. Check to see if an object is activated at this time
2. Check for a new update record
3. Check to see if object should be terminated
4. Update position

In this mode of operation an object can be displayed as soon as it becomes "active" in the simulation. The graphics display system utilizes the velocity

information and the frame update rate to determine the next location to display each object. Whenever a new location becomes current (based on simulation time) the velocity data from the new record is used. As long as the location records arrive before the display clock reaches the simulation time for the update, overshooting a position should not occur.

Table 4.3. Location Data Structure

| *Field* | *Description* |
|---|---|
| time | simulation time |
| x | position coordinate |
| y | position coordinate |
| z | position coordinate |
| Vx | Velocity Vector |
| Vy | Velocity Vector |
| Vz | Velocity Vector |
| heading | orientation angle |
| pitch | orientation angle |
| roll | orientation angle |
| Vheading | Velocity angle |
| Vpitch | Velocity angle |
| Vroll | Velocity angle |
| object | object number |

*4.3.3 File System.* The various files required to execute VISIT are listed in Table 4.4. A default parameter file is supplied that establishes the correct viewing parameters for the test datafile used throughout development.

Appendix B identifies the structure and content of the parameter file. A configuration file is also required that establishes the correct data for the current hardware configuration. The geometry description files currently available are listed in Appendix E.

Table 4.4. Data Files Required

| *File* | *Description* |
|---|---|
| configuration file | identifies spaceball configuration |
| parameter file | identifies initial viewing parameters |
| data file | simulation records datamode only |

## *4.4 User Interface*

The system is executed using the following command line.

```
VISIT -f parameter file
[-h remote host [-p remote program]]
[-n] ntsc mode
[-s] I/O server mode
[-c] I/O config file
```

The -f switch is used to specify the parameter file used for initialization. The -h option specifies the remote host used for the simulation processor. The -p option specifies the program that is executed on the simulation processor. The -n option allows the graphics system to use an ntsc monitor for the display. The -s option specifies that the external I/O device is read from the VEDS processor using the config file specified with the -c option. A menu-based script file is available that incorporates various default configurations.

*4.4.1 User Environment.* To provide a friendly user environment, many of the viewing parameters and display controls are available for modification by entering a single keystroke on the keyboard or by using the popup menu with the mouse. A list of keyboard functions and a list of functions provided with the mouse menu is included in Appendix A. A separate window is maintained in the upper-right corner of the display screen for on-line help. A menu of topics is displayed and allows the viewer to retrieve help on the keyboard, mouse, and spaceball functions.

Figure 4.1 shows a screen snapshot with the various viewing parameters displayed. The display of the parameters is user selectable and allows the viewer to monitor the various parameters as they are altered. Also the clock statistics are displayed in the upper-left corner of the window. The viewer can control the speed of the display clock and monitor the correlation between the display clock and the simulation clock in real-time.

*4.4.2 Viewing Control.* Throughout the execution of the simulation the viewer maintains considerable control over the viewer's display. Once the display has been set in motion, the viewer may pause the display, reset the simulation clock, step through the display a frame at a time, or establish a checkpoint. Viewer control is maintained by storing and keeping all update records for each object within memory. Resetting the display clock time is accomplished by relocating the current location pointer for each object within the data structure of location records.

The simulation clock may be reset to time 0 or to any time up to the current simulation clock time. A checkpoint can be set any time during the display of a simulation for any simulation time into the future. Upon reaching this time, the display will pause and allow the viewer to interact with the simulation objects. Once the viewer has finished interacting with the

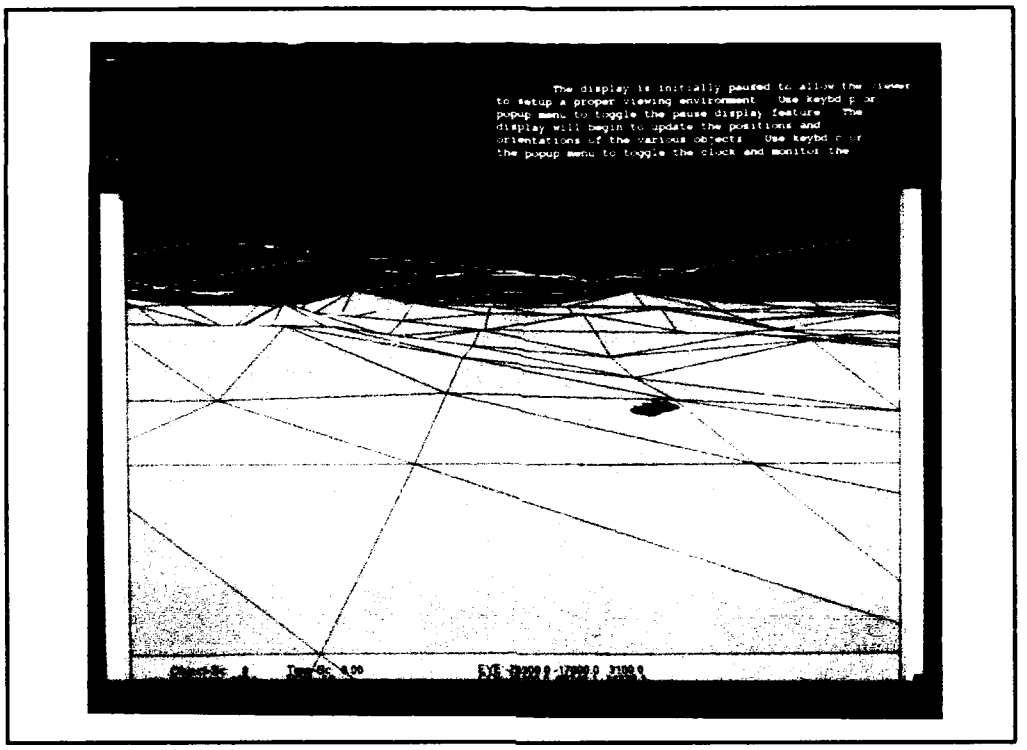

Figure 4.1. Screen Snapshot with Viewing Parameters Displayed

simulation the display may be set back in motion with a single keystroke.

Viewer interaction is provided with the CIS Dimension Six spaceball. The device provides six degrees of freedom and is used to select objects, relocate objects, and manipulate the viewing position. External I/O to this device requires considerable time and is a key factor in the performance of the interface. The device is made available to the system in two modes, server mode and local mode.

In server mode the data are received from the VEDS processor and in local mode the data are received from the graphics display system's serial port. Either way a reduction is seen in the frame update rate in most situations (see Table 5.1). The reduction is due to the system accessing the device for data each time through the display loop. To compensate, a flag is maintained

in the system to "pass over" reading the spaceball. This allows the viewer to control when the system accesses the device for data.

When the spaceball functions are required, the viewer presses the applicable keyboard key and the system will execute the code that accesses the device for data. When using the spaceball, the display is usually paused, and no concern exists for how fast the frame buffer is updated because the simulation objects are suspended at their current location. Once the viewer has finished using the spaceball, the viewer can toggle off access to the spaceball before releasing the display in motion. A detailed discussion of the spaceball operation can be found in Appendix A.

### *4.5 Graphical Issues*

*4.5.1 Z-Buffer.* Using the hardware Z-Buffer on the Iris 4D/85GT presented some unusual problems during the course of development. The initial displays created showed polygon tearing in the geometric shapes used to represent the simulation objects. The initial evaluation determined that there was a hardware inconsistency in what appeared to be the Z-Buffer. Later analysis by Silicon Graphics determined that the model 4D/85GT possessed a different design in the hardware that is used for transforming the geometric shapes to the display screen. During this analysis, software development was ported to a different workstation, yet the software anomalies associated with the Z-Buffer did not go away.

Upon detailed analysis, the perspective and lsetdepth functions of the graphics library, provided with the system were used incorrectly. The lsetdepth function takes two parameters: near and far, and the perspective function takes four parameters: field of view, aspect ratio, near clipping plane, and far clipping plane (17). The problems encountered were the result of incorrectly setting the parameters to these functions. Equations 1 and 2 depict the

interrelationships between the parameters of these two functions. Lsetdepth, by default has the near parameter set to the minimum value that can be stored into the Z-buffer and far set to the maximum value. During the rendering process the z screen coordinate ($0 \leq z_{screen} \leq 1$), is computed from the perspective transformation and mapped into an integer value ($near \leq z_{viewport} \leq far$). The computed value is then compared to what currently exists in the Z-Buffer.

$$z_{screen} = \left[\frac{far + near}{far - near} + \frac{2 \cdot far \cdot near}{z_{eye}(far - near)}\right]\left[\frac{far_{vp} - near_{vp}}{2}\right] + \frac{far_{vp} + near_{vp}}{2} \tag{4.1}$$

$$z_{eye} = \frac{\dfrac{far \cdot near \cdot (far_{vp} - near_{vp})}{(far - near)}}{z_{screen} - \dfrac{(far + near)(far_{vp} - near_{vp})}{2(far - near)} - \dfrac{far_{vp} + near_{vp}}{2}} \tag{4.2}$$

When this comparison involves values approaching the two opposite ends of the range of near and far, a "wrap around" condition occurs due to the hardware used for this comparison on the 4D/85GT. In essence, when the hardware should select the object nearest to the viewer, the hardware selects the object farthest away. This results in polygon tearing: portions of one polygon being replaced with portions of another incorrectly at overlapping surfaces. To compensate for this condition, the far parameter should be set to a value less than the default and/or the near parameter should be set to a value greater than the default.

Also related to the polygon tearing was the parameter specifications to the perspective function. As previously stated, the z value specifies the distance from a given pixel to the eye point, utilizing 24 bits of accuracy for a maximum distance of $2^{23}$ or 8,388,608. The relationship between the z

value computed and the distance is linear only when using the orthographic projection. When using the perspective projection this relationship is non-linear and is dependent upon the ratio of the far clipping plane to the near clipping plane. As this ratio increases, the degree of non-linearity increases. As a reference, ratios greater than 1000 show effects of polygon tearing (1:32). During the initial development, a near value of .01 and a far value of 4,000,000 were used, which resulted in a ratio far exceeding the recommended maximum.

To achieve an acceptable display, the ratio had to be reduced. The easiest way to reduce the ratio and yet maintain an acceptable image was to increase the near clipping plane. After experimentation with various values for both functions, an acceptable display quality was achieved from most viewing positions. However, when the viewer's eye position was placed within an object and moving about in the display, objects that approached the viewing position were being clipped away. To compensate, the perspective function was called whenever entering this mode of viewing and the near clipping plane was reduced to 10. Upon exiting this mode the near plane was reset.

*4.5.2 Viewing.* The viewer has complete control over what position to use as a viewing position of the simulation. The viewer may establish a viewpoint at a static location and watch the entire scenario from that position. However, the interface allows both keyboard control and spaceball control to maneuver the viewing position. This enables the viewer to position the viewpoint virtually anywhere that is within the hardware limits of the display system.

Another useful feature is one that allows the viewer to position the viewpoint within one of the simulation objects. This provides the viewer with the ability to watch the simulation from that object's moving perspective.

This view ranges from the slow ground-level perspective of a tank to the view obtained from a missile launched from an aircraft. Figure 4.2 is a screen snapshot depicting a view of the test simulation from one of the aircraft.

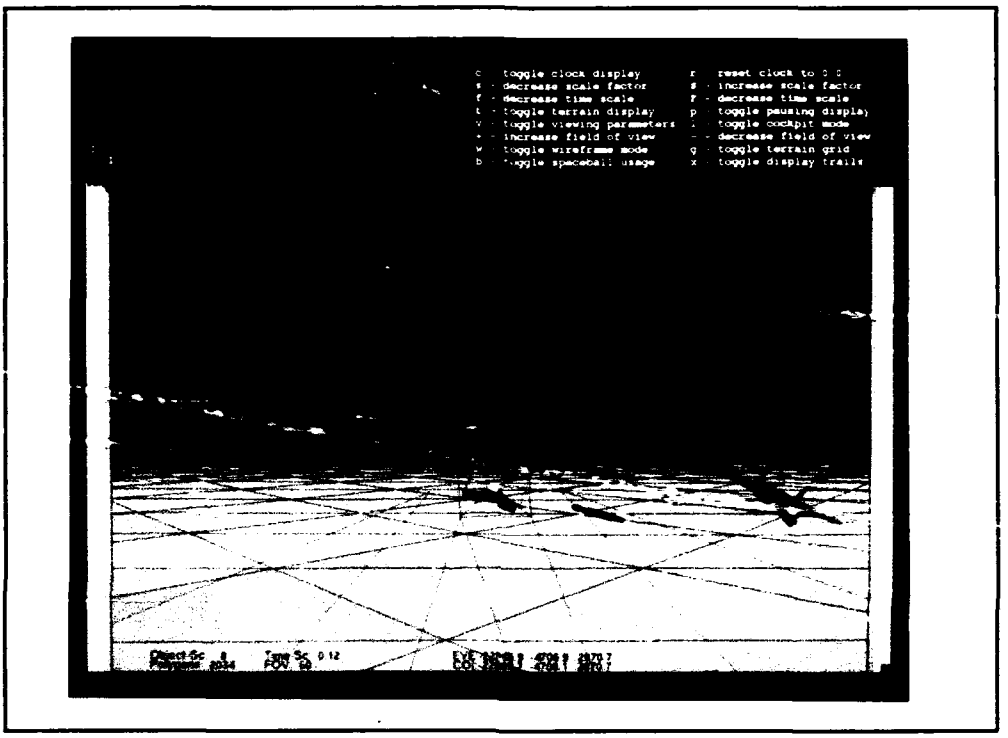

Figure 4.2. Screen Snapshot of a Simulation Object's View

## *4.6 Summary*

The network communications proved to be a big challenge throughout the development of the software interface and the software tool Simtoge. The spaceball communication connections between the various systems also presented some difficult times. However, the prototyping method of software development worked very well with this thesis. The decomposition of the problem into the three major areas provided a manageable method of developing the software. The Iris 4D/85GT system provides a very flexible software development environment that worked well for this effort. The subnet layout

of the LAN used within AFIT was very well organized and provided a good testbed for the network communications.

# V. Results and Recommendations

## 5.1 Introduction

Overall, the required objectives were achieved. Fast, efficient network communications are maintained between the simulation processor and the graphics display system. Realistic animation is achieved in most situations, but is highly dependent upon the complexity of the scene. The viewer maintains complete control over the execution and display of the simulation. Existing simulation output, once converted into a format-specific datafile, can be displayed utilizing an incorporated software tool that executes on the simulation processor.

## 5.2 Results

A test scenario was constructed to thoroughly test the various aspects of the software interface. This scenario contained 35 simulation objects. The objects ranged from missiles that utilize 25 polygons in their description to aircraft that contain over 100 polygons in their description. The scenario had an average of 2,000 polygons within each frame. There were 645 location records transferred from the simulation processor to the graphics display system. The vast majority of these records were for four aircraft that were in existence throughout the entire simulation. Figure 5.1 shows a user viewing the test scenario using the spaceball to manipulate the display.

Table 5.1 contains the frame update rates achieved using this scenario under various conditions. The table identifies what factors were critical in determining the frame update rate. The terrain was a simple 3-dimensional description using 1,152 3-sided polygons. A grid was formed by creating closed-line polygons using the same description as the terrain. Combined with

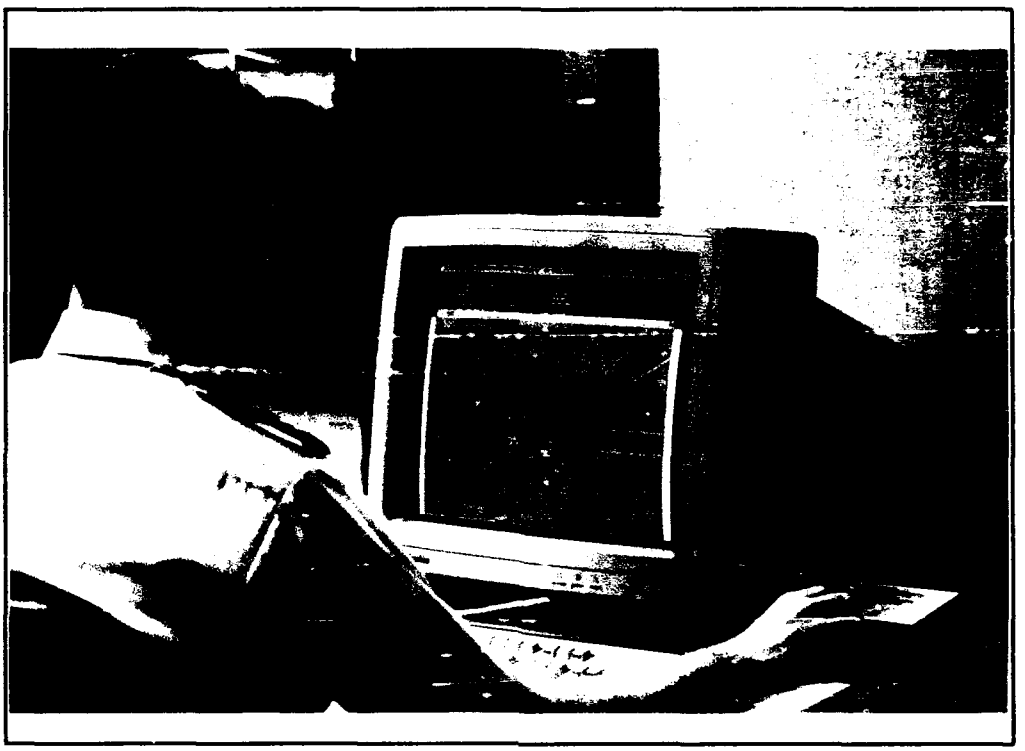

Figure 5.1. Viewer Interacting with VISIT

the simulation objects a frame contained 3,150 polygons on average when the terrain features and grid were displayed.

Another key factor was the display of the clock statistics. This entailed displaying raster font character strings on the screen and querying the system clock during the display loop. As can be seen by the table, when the entire window contains terrain features there is little performance degregation displaying the grid and clock statistics. This is due in part to the system having to paint each pixel in the frame buffer each time through the display loop. When the terrain features are not displayed, the frame update rate is almost tripled, but a big degregation is seen when the clock statistics are displayed.

The other factor was the spaceball access time, which was discussed

previously in Section 4.4.2. The table shows little performance loss when the terrain occupies the entire window and the spaceball is accessed in server mode. However, in local mode the spaceball reduces the frame update rate severely.

### *5.3 Evaluations*

The VPL dataglove was utilized during initial development as an input device for this interface. However, the correlation of hand movements and rotations to a virtual hand displayed on the screen could not be accurately computed. In retrospect, the approach of computing a ratio of physical movement to virtual movement with a large contrast in size between the two was incorrect. A different approach based on the functionality of the spaceball may warrant exploration.

A hand gesture could be used to indicate which direction and axis to move the viewing position or center of interest. Then using some type of throttle gesture, the viewer can control the speed of movement. In this fashion the viewer can move relatively fast across large spans of the display and relatively slowly when narrowing in on a target area of the screen. This approach is quite similar to how the spaceball uses the amount of torque applied to the ball in a given direction to indicate the amount of movement.

### *5.4 Recommendations for Further Research*

A hierarchical approach to simulation objects is desired when displaying large simulations. The hierarchy of the simulation should be representable, in some fashion, on the display screen. Also, communications with a simulation executing on a parallel processor system in real-time should be examined. The interface should be able to handle the display requirements as long as the data records reach the display system before they are required. However the

Table 5.1. Frame Update Rates (using 1200 x 800 window)

| *Terrain Features* | *Grid* | *Clock* | *Spaceball* | *Frames/sec* |
|---|---|---|---|---|
| Full Window | off | off | off | 11.9 |
| | off | on | off | 9.9 |
| | on | off | off | 10.0 |
| | on | on | off | 10.0 |
| | off | off | Server | 12.1 |
| | off | on | Server | 10.0 |
| | on | off | Server | 10.1 |
| | on | on | Server | 8.7 |
| | off | off | Local | 8.7 |
| | off | on | Local | 7.3 |
| | on | off | Local | 7.3 |
| | on | on | Local | 6.5 |
| Half Window | off | off | off | 12.0 |
| | off | on | off | 12.5 |
| | on | off | off | 10.0 |
| | on | on | off | 9.8 |
| | off | off | Server | 14.7 |
| | off | on | Server | 11.6 |
| | on | off | Server | 9.9 |
| | on | on | Server | 9.0 |
| | off | off | Local | 9.9 |
| | off | on | Local | 8.0 |
| | on | off | Local | 7.9 |
| | on | on | Local | 6.8 |
| Not Displayed | off | off | off | 29.3 |
| | off | on | off | 20.5 |
| | off | off | Server | 29.3 |
| | off | on | Server | 19.4 |
| | off | off | Local | 13.9 |
| | off | on | Local | 11.4 |

current version maintains the location records in system memory and does not write them to disk. A very large simulation may require some type of intermediate disk saving capability.

## *5.5 Conclusions*

The viewer has the ability with this interface to completely control a simulation's execution and display. This provides the necessary capabilities so often needed by developers and users. The interface has the ability to see the simulation as it executes knowing immediately if the simulation is valid, knowing immediately if the input data was in the correct format, knowing immediately if results were obtained. It has the ability for a developer to validate the requirements with a user by showing instead of explaining. It contains the ability for a group to see the execution and control of a simulation, which may or may not be executing in the local environment. Finally, the interface has the ability to display a simulation that executes on hardware with no graphics display capability.

## *5.6 Summary*

The resulting interface provided a very efficient, fast, and reliable medium to transfer data between a simulation processor and the graphics display system. The graphics system provided real-time display capabilities which maintained a reasonable frame update rate. User interaction was provided in various forms that allowed the viewer to interactively control and manipulate the simulation.

# Appendix A. *VISIT User's Guide*

*Introduction*

VISIT is an interactive simulation display system that permits the graphical display of a simulation that is concurrently executing on another system. Display data is received by the graphics display system and commands are sent to the simulation processor via ethernet communication utilizing the TCP/IP protocol.

The following sections explain how to setup and operate VISIT. In addition, the various file formats are explained in detail.

*Preparing VISIT for Use*

Before VISIT can be used various initialization tasks must be accomplished in the proper sequence. Figure A.1 shows a block diagram of the complete VISIT system.

The following procedure should be used when executing the display of a simulation.

1. Begin by logging into the Silicon Graphics Iris 4D/85GT
2. Ensure appropriate files are present on the systems used (see Table A.1).
3. Type VISIT <cr> to execute the main menu
4. Select the appropriate configuration from the menu.

Once the message to start displaying the simulation is received, the graphics system will begin the display. The display initially appears in pause mode to allow the viewer to manipulate the viewing parameters or to change the display control characteristics. The animation of the simulation will begin

after a keyboard p is pressed. To exit the system, the viewer enters a keyboard q. An on-line help facility is available by typing HELP <cr> in the help window.

*Interactive Input*

The parameter file provides the various initial parameters. However the system provides the flexibility to interactively modify the various viewing parameters and the simulation objects themselves. Keyboard functions, a mouse menu, and the spaceball functions provide these capabilities.

*Keyboard & Mouse.* The keyboard provides access to virtually all viewing parameters and allows the user to interactively modify the simulation environment. The list of keyboard functions is displayed in Table A.2.

The mouse provides a single popup menu by clicking on the right mouse button. The options are selected by clicking the right mouse button when the applicable option is highlighted. To cancel **before** selecting an option, click on the menu title and you will return to the display. The popup menu functions are listed in Table A.3.

*CIS Dimension Six Trackball.* THE CIS Dimension Six force-torque ball (shown in Figure A.2) is a six degree of freedom input device combining the

Table A.1. Data Files Required

| *File* | *Description* |
|---|---|
| configuration file | identifies spaceball configuration |
| parameter file | identifies initial viewing parameters |
| data file | simulation records datamode only |

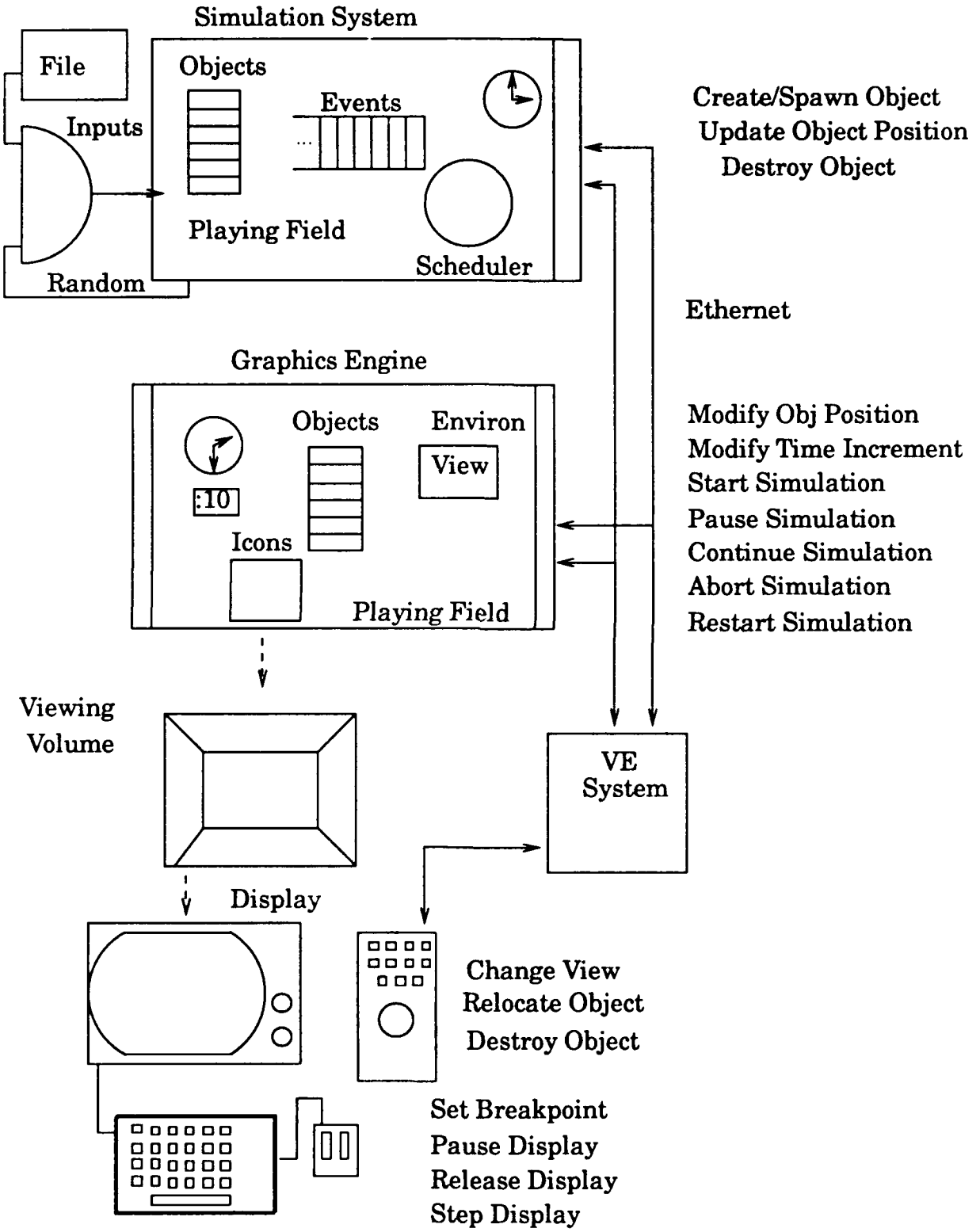


Figure A.1. System Architecture Layout

functionality of a joystick, a button box, and a dial box (7). The ball allows rotations about the three primary axes and translations in three directions. Force sensors inside the ball register the amount of force and torque applied to the ball and send this information over an RS-232 interface to the host.

The device has eight function buttons that provide user input capability (see Table A.4). There is also a set of three buttons on the trackball that allow the user to change from translations to rotations. The force applied to the ball is sent to the host and read into a data structure. The status of the three control buttons determines how that data structure is filled. The trackball is used to both modify the viewing parameters and alter the location and/or existence of simulation objects.

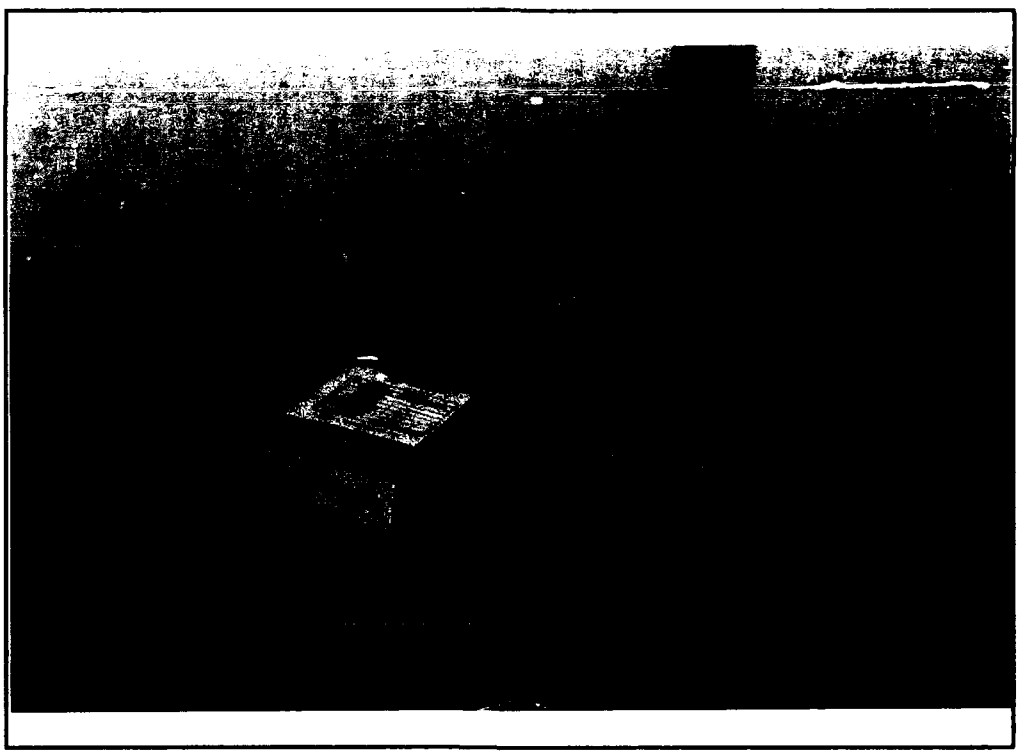

Figure A.2. CIS Dimension Six Force-Torque Ball

*Viewing Mode.* In viewing mode the user is allowed to either zoom in or out, or pan along one of the three primary axes. To zoom, the user depresses a function button and then applies force to the ball in the direction specified in Table A.5 to achieve the desired change in the display. As the system reads the amount and direction of force, the display is updated. Once a position is reached and the user is satisfied, a function button is depressed to exit viewing mode.

To pan the display, the user depresses a function button and then applies force to the ball in the direction in which the display is to pan. This system uses the left-hand coordinate system, so applying force to the right will pan along the positive x axis, pushing forward on the ball will pan along the positive y axis, and pulling up on the ball will pan along the positive z axis. Applying force in the opposite direction for either axis will pan along the respective negative axis. Again, once a suitable position is reached, a function button is depressed to exit viewing mode (see Table A.4).

*Picking Mode.* The user can maneuver a cursor anywhere within the current display screen and 'pick' an object, move that object with the trackball, and 'drop' the object at the new location. The user may also delete an object from the current display once it has been picked. If the object that the user wants to pick is not currently within the viewing window, the user can enter viewing mode and reposition the window so that the desired object is viewable.

The user enters picking mode by depressing a function key on the trackball. This causes the system to display a wireframe cube in the center of the viewing window. The user then maneuvers the cube by applying force in the appropriate direction towards the object that is to be picked (see Table A.5). When the object appears within the cube or near it, the user depresses another

function key. This triggers the software system to calculate the distance from the cube to all viewable objects within the simulation. The closest object to the cube is then displayed in wireframe mode for visual feedback to the user. At this point the user has three options.

If the object that was calculated by the system to be nearest to the cube is *not* the object that the user wanted selected, a function key is available to 'unpick' the object. This will reset the display mode of the object back to shaded and display the cube again. The user may then maneuver the cube closer to the desired object and pick it again.

A second option after the object is picked is deletion. The user can depress a function key that will cause that object to be removed from the current list of displayable objects. This will also trigger the system to send a network message to the simulation system that an object was deleted from the simulation.

A final option is to relocate the object. Once the object has been picked, the user may apply force to the ball and thus relocate the object within the viewing window. The user can also reorient the object by applying force to the ball in any one of three rotation directions. The user can switch between translations and rotations by depressing the applicable control button on the device. Once a suitable position and orientation has been achieved, a function key is depressed to 'drop' the object at its current displayed location. This also triggers the system to send a network message to the simulation that the location of an object has been altered.

Table A.2. Keyboard Functions

| *Key* | *Function* |
|---|---|
| + | increase field of view |
| – | decrease field of view |
| b | toggle spaceball |
| c | toggle clock display |
| f | decrease time scale |
| F | decrease time scale |
| g | toggle terrain grid |
| l | toggle cockpit mode |
| p | toggle pausing display |
| r | reset clock to 0.0 |
| s | decrease scale factor |
| S | increase scale factor |
| t | toggle terrain display |
| v | toggle viewing parameters |
| w | toggle wireframe mode |
| x | toggle display trails |

Table A.3. Mouse Menu Functions

| *Option* | *Function* |
|---|---|
| Toggle Clock | toggle clock display |
| Toggle Parameters | toggle viewing statistics |
| Toggle Terrain | toggle terrain display |
| Toggle Pause | toggle pausing display |
| Simulation Control | submenu to control simulation |
| Clock Control | submenu to control display clock |
| Pick Object | pick object to view from |
| Change Terrain | change terrain file |
| | |
| Pause | send msg to pause simulation |
| Continue | send msg to continue simulation |
| Stop | send msg to stop simulation |
| Abort | send msg to abort simulation |
| Restart | send msg to restart simulation |
| | |
| Reset Clock | set clock to 0.0 |
| Set Clock | set clock to time n |
| Set Breakpoint | set breakpoint to pause display |
| Step Display | step through display by frames |

Table A.4. Trackball Function Keys

| *Key* | *Function* | *Comments* |
|---|---|---|
| 1 | Display Cube | Begin picking mode |
| 2 | Pick Object | |
| 3 | Cancel Pick | |
| 4 | Drop Object | |
| 5 | Activate Zoom | |
| 6 | Delete Object | |
| 7 | Activate Pan | |
| 8 | Deactivate | Terminates movement |

Table A.5. Trackball Translations

| *Key* | *PAN Function* | *ZOOM/MOVE Function* |
|---|---|---|
| slide ball forward | pan forward | move entity along +y |
| slide ball backward | pan backward | move entity along -y |
| slide ball right | pan eastward | move entity along +x |
| slide ball left | pan westward | move entity along -x |
| slide ball upward | pan upward | move entity along +z |
| slide ball downward | pan downward | move entity along -z |

# Appendix B. *Parameter File Format*

The VISIT parameter file provides a method for initializing the viewing parameters. The file is arranged so that only one parameter keyword and its associated value(s) can be on a single line. The order in which the parameters appear within the file is not important.

The file syntax is shown in Table B.1. In the table, literal symbols appear as contiguous strings of alphabetic characters. Substitutable symbols appear in angle brackets "<" and ">". Optional information appears within square brackets "[" and "]". C-style comments (`/* */`) and `#include` and `#define` directives may be contained within a parameter file.

Table B.1. VISIT Parameter File Format

| *Keywords* | *Comments* |
|---|---|
| COI <x> <y> <z> | Center of Interest Coordinates |
| EYE <x> <y> <z> | Eye Position Coordinates |
| FOV <angle> | Field of View |
| FUR <rate> | Frame Update Rate |
| IDIR <path> | ICON Directory |
| MTT <time> | Maximum Simulation Time |
| RA <angle> | Rotation Angle Amount |
| SF <factor> | Object Scaling Factor |
| SS <factor> | Movement Step Size |
| TER <x> <y> <z> | Terrain offset |
| TF <factor> | Time Scaling Factor |
| TFILE <filename> | Terrain Geometry Description |
| CSCALE <factor> | Coordinate Scaling Factor |
| MANGLE <mode> | Angle Measurement in CW Direction |
| NPLANE <distance> | Near Clipping Plane |
| FPLANE <distance> | Far Clipping Plane |

# Appendix C. *Datamode File Format*

The datafile is composed of records of several types. Each record type contains fields in a specific format. The number of fields in a record is different for each record type. In all cases, the first field contains an integer which defines the record type.

*Types*

*Icon Assignment.* Assigns an icon index to a viewable object.

Table C.1. Record Type 30

| *Field* | *Description* | *Data Type* |
|---|---|---|
| 30 | Record Type | integer |
| O | Object Index Number | integer |
| I | Icon Index Number | integer |

```
Example: 30 3 8
```

This examples indicates simulation object number 3 is assigned to icon index 30. Simulation Object numbers must begin with 1 and be sequential.

*Object Location.* Contains position and orientation data for a viewable object. The position and velocity values have a maximum width of eleven characters. This width is inclusive of a minus sign and a decimal position. The angles are measured according to the right-hand rule, which is as follows: as you look down the positive rotation axis to the origin, positive rotation is counterclockwise (17:7-17).

Table C.2. Record Type 31

| *Field* | *Description* | *Data Type* |
|---|---|---|
| 31 | Record Type | integer |
| O | Object Index Number | integer |
| T | Time (seconds) | float |
| X | X - position (meters) | float |
| Y | Y - position (meters) | float |
| Z | Z - position (meters) | float |
| VX | velocity in x (meters/sec) | float |
| VY | velocity in y (meters/sec) | float |
| VZ | velocity in z (meters/sec) | float |
| H | Heading (degrees) | float |
| P | Pitch (degrees) | float |
| R | Roll (degrees) | float |
| VH | change in Heading (degrees/sec) | float |
| VP | change in Pitch (degrees/sec) | float |
| VR | change in Roll (degrees/sec) | float |

```
Example: 31 2 2.5 1000 500 -20 1.2 2.4 -.3 30.0 10.0 -90.0 0.5 5.0 -1.(
```

This examples indicates that object number 2 at simulation time 2.5 is positioned at 1000, 500, -20. The object has a heading of 30 degrees, a pitch of 10 degrees, and a roll of -90 degrees. The velocity components are 1.2, 2.4, and -.3 for position and 0.5, 5.0, and -1.0 for orientation.

*Icon Identification.* Identifies an icon by index and geometry description filename.

Table C.3. Record Type 32

| *Field* | *Description* | *Data Type* |
|---|---|---|
| 32 | Record Type | integer |
| I | Icon Index Number | integer |
| F | Icon Filename | character string |

```
Example: 32 8 mig1
```

This example indicates that icon index number 8 is associated with the geometry file mig1. Icon numbers are determined freely by the user.

*Object Termination.* Identifies when an object is to be terminated. This is the time at which the display system stops displaying the object.

Table C.4. Record Type 33

| *Field* | *Description* | *Data Type* |
|---|---|---|
| 33 | Record Type | integer |
| O | Object Index Number | integer |
| T | Termination Time | float |

```
Example: 33 3 115.5
```

This examples indicates that object number 3 will be no longer displayed at simulation time 115.5.

*Start Display.* Indicates all icons and the initial starting positions have been identified and sent to the graphics engine. The graphics engine can begin displaying the simulation.

Table C.5. Record Type 50

| *Field* | *Description* | *Data Type* |
|---|---|---|
| 50 | Record Type | integer |

*Reset Display.* Indicates to the graphics display system that the simulation was restarted and will begin execution. The graphics display system will pause until a START DISPLAY is received.

Table C.6. Record Type 52

| *Field* | *Description* | *Data Type* |
|---|---|---|
| 52 | Record Type | integer |

*End of Simulation.* Indicates the end of the simulation. This will be the last line within the datafile that is read.

Table C.7. Record Type 86

| *Field* | *Description* | *Data Type* |
|---|---|---|
| 86 | Record Type | integer |
| T | Termination time | float |

```
Example: 86 245.0
```

This example indicates that the simulation display is to stop at simulation time 245.0.

*Ordering*

All icon identifications (type 32) must occur before any other type of record in the datafile. Each viewable object must be associated with an icon (type 30) before a location record (type 31) for that object can occur in the datafile.

# Appendix D. *Message Packet Descriptions*

*Introduction*

The fields of the message packet used by this interface are shown in Table D.1. The icon field is only used by the icon assignment and icon identification type packets; all other packets avoid this field. The buf field contains type-specific data and is identified specifically for each type packet.

This message packet structure was adapted from an existing network communications package. This package allowed communication between the 4D/85GT system and a parallel processing system; however, it relied on the message header fields be of type short. This restriction prevented an easy way of storing the simulation time, (a float type), within the message header, and that is why the simulation time is stored in the buffer instead of being a field within the message header.

*Data Types*

Table D.1. Message Packet Structure

| *Field* | *Description* |
|---|---|
| request | Request Type |
| object | Object Index Number |
| icon | Icon Index Number |
| bufIth | Length of Buffer |
| buf | Data |

*Type 30 - Icon Assignment.* Assigns an icon index to a viewable object. The buffer contains 0 bytes of data for this record. Object numbers must begin with 1 and be sequential.

*Type 31 - Object Location.* Contains position and orientation data for a viewable object. This record type should not be sent until after the icon assignment packet has been sent for the object number in this packet. The buffer contents for this record are the following:

simulation time,x,y,z coordinates, x,y,z velocity vector coordinates, heading, pitch, roll, heading change, pitch change, and roll change.

All values are separated by one space character.

*Type 32 - Icon Identification.* Identifies an icon by index and geometry description file that is used by the display system to represent the object. The buffer contains the filename only. This filename is concatenated to the path name provided in the parameter file. This packet type must be sent for each icon used, but before any icon assignment is sent for that particular icon.

*Type 33 - Object Termination.* Identifies the simulation time in which an object is terminated. The buffer contains the simulation time at which termination occurs.

*Viewer Interaction Types*

These message packet types are used when the viewer interacts with the display of the simulation using the spaceball device. These commands have no effect in the data mode of operation.

*Type 34 - Move Object.* This message packet is used to send the new location of an object back to the simulation. The contents of the buffer are:

simulation time, x, y, z position, roll, pitch, and heading.

*Type 35 - Destroy Object.* This packet is used to send the simulation time at which the user interactively deleted an object. The simulation time is placed into the buffer.

*Simulation Control Types*

The simulation control is quite different from the display control capability of the system. When the user selects to pause the display only the display and object position updating pause; no message packet is sent to the simulation processor. The simulation control packets are for communication with the simulation processor. The current simulation time is loaded into the buffer before the packet is sent for all packets of this type.

*Type 40 - Pause Simulation.* The display system sends this message to the simulation system when it becomes necessary to pause the simulation. Currently, this condition never occurs internally and is allowed only as a user selected option.

*Type 41 - Continue Simulation.* The display system sends this message to the simulation system when it becomes necessary to continue the simulation. Currently, this condition never occurs internally and is allowed only as a user selected option.

*Type 42 - Stop Simulation.* The display system sends this message to the simulation system when it becomes necessary to stop the simulation. Currently, this condition never occurs internally and is allowed only as a user selected option.

*Type 43 - Abort Simulation.* The display system sends this message to the simulation system when it becomes necessary to abort the simulation. Currently, this condition never occurs internally and is allowed only as a user selected option.

*Type 44 - Restart Simulation.* The display system sends this message to the simulation system when it becomes necessary to restart the simulation. Currently, this condition never occurs internally and is allowed only as a user selected option. This is the only simulation control message packet that the simulation postprocessor, Simtoge, uses. The other five packets are read and an acknowledgment is displayed in the console window; no action is taken.

*Type 45 - Save State.* The display system sends this message packet when a user determined breakpoint is reached.

*Type 25 - Quit Communications.* This packet is sent by the graphics display system as a house-cleaning function before termination of the interface. The buffer contains the current simulation time.

*Status Types*

*Type 50 - Start Display.* Instructs the graphics display system to begin displaying the simulation. At this point the display system will allow user interaction and control. This packet should not be sent until all icon identification and icon assignment packets have been sent for the objects that are known at simulation time zero. The buffer and object index field will be empty.

*Type 52 - Reset Display.* Instructs the graphics display system that all current data is no longer valid and the simulation will restart. The simulation must send a Start Display packet to allow the display system to continue. The

buffer and object index field will be empty.

*Type 86 - Datafile Complete.* Instructs the graphics display system that reading of the datafile is complete. The buffer and object index field will be empty.

*Modifications*

As stated in the introduction this message packet structure is based upon a communications package received from Oakridge National Laboratory. The modifications that were made are noted here because it's the only place that they fit in.

First, the original package relied on a global visibility for the network socket file descriptor and message packet structure. That could not occur in the interface. It was necessary to make these two entities parameters for all modules that required them.

Second, several new request types were added to the original package. Basically all request types identified in this appendix are new and are identified in the header file /veds/oakridge/include/AFITcom.h. All previous request types were maintained. A subset of the original collection of modules was combined into one file, identified as /veds/oakridge/lib/simutils.c. This file is linked in by both the interface VISIT and the simulation postprocessor Simtoge.

# Appendix E. *AFIT Geometry File Format*

The AFIT geometry file provides a method for describing three dimensional objects composed of planar polygons, and for specifying attributes such as color and a shading model. The file is organized in a position dependent manner such that the position of a line within the file (its "line number") determines the class of information a line may contain.

The file syntax is shown in Table E.1. In the table, literal symbols appear as contiguous strings of alphabetic characters. Substitutable symbols appear in angle brackets "<" and ">". Optional information appears within square brackets "[" and "]". C-style comments (`/* */`) and `#include` and `#define` directives may be contained within an AFIT geometry file.

A list of available geometry descriptions using this format is shown in Table E.2.

Table E.1. AFIT Geometry File Format

| *Line* | *Keywords* | *Comments* |
|---|---|---|
| 1 | None | Up to 1024 characters |
| 2 | [ccw] [cw] [purge] [nopurge] | Geometry Parameters |
| 3+ | points <# of points> | Object component and |
| | patches <# of patches> | attribute counts |
| | [attributes <# of attributes>] | |
| | [textures <# of textures>] | |
| 4+ | <x> <y> <z> | Vertex Lines |
| | [normal <i> <j> <k>] | |
| | [color <r> <g> <b> | |
| | [tindex <u> <v>] | |
| 5+ | <n> <pt 1>...<pt n> | Polygon/Patch Lines |
| | [attribute <n>] | |
| | [texture <n>] | |
| | [type {PLAIN, COLOR, TEXTURE}] | |
| 6+ | [shading {FLAT, GOURAUD, PHONG}] | Attribute Lines |
| | [reflectance {FLAT, PHONG, COOK}] | |
| | [kd <n>] | |
| | [ks <n>] | |
| | [n <n>] | |
| | [opacity <n>] | |
| | [color <r> <g> <b>] | |
| | [m <n>] | |
| | [material <filename>] | |
| 7+ | filename | Texture Map Filename |

Table E.2. Available Geometry Files

| *Filename* | *Polygons* |
|---|---|
| f18 | 29 |
| a10 | 12 |
| f14 | 105 |
| mig1 | 101 |
| missile | 25 |
| tank | 20 |
| truck | 14 |

# Appendix F. *Simulation Developer's Guide*

*Introduction*

This appendix is meant to assist anyone who will be using the interface in sim mode and has to design into a simulation the various "hooks" used by this interface. There are various features that should be incorporated into a simulation in order to utilize the maximum functionality of this interface. However, very little extra work should be required to just view a simulation; the majority of the extra work is required for viewer interaction and control.

*Data Representations*

The objects within the simulation are represented graphically on the display system by what is referred to as icons. To represent an object on the display screen the graphics display system must know what to use for it's representation. Table E.2 lists the currently available icons. These icons are described using polygonal descriptions in the file format specified in Appendix E.

All objects are described with their center oriented at the origin of the world coordinate system. The front of the object is pointed away from the origin down the positive y axis. The initial orientation of the object is zero for all rotation angles. All objects are measured in meters and are externally scalable to any size. The interface allows the viewer to control the object scale size, but that value applies to all objects known to the display system.

The orientation angles are measured according to the right-hand rule, which is as follows: as you look down the positive rotation axis to the origin, positive rotation is counterclockwise. The interface reads a parameter at initialization time that may be used to indicate that the angle measurements

are the reverse of this rule. The important thing to remember is be consistent, either all angles are counterclockwise positive or clockwise positive.

The position of objects are specified using the 3-dimensional right-handed cartesian coordinate system. The units used are of no importance to the display system except for consistency. Whatever unit of measurement is used for positioning, must be used throughout. If you use meters to describe the objects and miles to indicate their positions, an erroneous display will occur. Also, there is a coordinate scaling parameter in the parameter file that may be used to indicate to the display system to scale all coordinates by this factor.

When using a unit of measure other than meters, the object descriptions may be scaled within the interface by the viewer. A keyboard function is available to the viewer to scale up or down by a factor of 2 all object representations. This is a global factor that applies to all objects and may not be applied to individual objects.

Object positions and orientations are sent to the graphics display system whenever a position or orientation changes. The packet that is sent requires the position coordinates, orientation angles, and velocity vector coordinates for the position and orientation. This information is used to determine the objects movement within the simulation. These records must be sent to the display system in simulation time order for any given object.

*Communications*

Appendix D describes the various message packets that are sent back and forth between the two systems. Appendix C describes the format of the datafile used for data mode operation by the simulation postprocessor, Sim'oge. Simple playback viewing of a simulation can be accomplished by creating a datafile in the format specified and using the data mode of operation.

However, to utilize the full functionality of the interface direct communications between the simulation processor and the graphics display system is required in real-time.

The necessary modules required for communications with the display system are available in the file /veds/oakridge/lib/simutils.c. This file contains the network communications modules used by the interface and the simulation postprocessor, Simtoge. The modules also contain the message packet structure that is used to transfer the data between the systems.

Many simulation control capabilities are provided by the interface that rely on the simulation processor to perform some function. In data mode these message packets are built and sent to Simtoge and an acknowledgment message is displayed in the console window. The viewer will have the ability to send the simulation a message to pause, stop, continue, restart, or abort the simulation. Also, the display system will send a message to the simulation processor when the display system reaches a viewer established breakpoint.

To assist the simulation in communicating with the display system, display status messages exist. The simulation must send a packet to the graphics display system when it is ready for the display system to begin displaying. This allows the simulation to perform all of it's initialization, identify it's objects, and begin it's execution before the display system begins.

It is important to remember that the display system will continue to display and update the simulation objects based on what is currently known. This requires the simulation processor to keep ahead of the graphics display system with respect to simulation time. The viewer may control this by pausing the display. Pausing the display pauses the display of the simulation at the current simulation time. In the background communication will continue between the two systems, thus allowing the simulation processor to continue sending data.

Finally, when the viewer exits the interface system a message packet is sent to the simulation processor to terminate. This is a send-only request, no return message will be accepted.

*Viewer Control & Interaction*

The viewer also has the ability to relocate and delete objects that are currently known to the graphics display system. Packets will be sent to the simulation processor when either of these actions are taken by the viewer. This will require some action be taken by the simulation. When an object is deleted by either a termination record from the simulation or a delete object from the viewer, the object number may not be reused.

*Summary*

The information presented in this appendix is intended to be a supplement to the other portions of this document. It is highly recommended that a developer read all appendices to this document before attempting to develop a simulation that will require this interface. Expansion of this interface to support parallel simulation applications should require minimal effort.

# Appendix G. *VISIT Man Page*

The following page contains a Unix manual page for the software interface.

## NAME

VISIT - Visual Interactive Simulation Interface Tool

## SYNOPSIS

**VISIT** **-f** *parameter* file [ **-h** *remote* host **-p** *remote* program ]
[ **-s** ] [ **-c** *config* file ] [ **-n** ]

## DESCRIPTION

*VISIT* is the Visual Interactive Simulation Interface Tool. It provides a graphical display of a simulation that is executing concurrently on another processor. The interface provides the user with the ability to control and interact with the real-time display of the simulation execution. A separate mode is available to display the output from a previous execution.

The initial operation of the interface is controlled by a parameter file. This file contains the initial value for various viewing parameters and graphical configuration variables.

The operation of the spaceball is controlled through a configuration file. This file specifies which input device is connected, to which port, and at what baud rate.

By default, the configuration file *VEdefault.config* is used for configuration information. This may be overridden using the **-c** option.

The format of the configuration file is shown by the following line.

spaceball /dev/ttyXX 19200

## SEE ALSO

server(LOCAL)

## AUTHOR

Bill DeRouchey

# *Bibliography*


1. Akeley, Kurt. "The Hidden Charms of Z-Buffer," *IRIS Universe, 11* (1990).
2. Bell, Peter C. and Robert M. O'Keefe. "Visual Interactive Simulation – History, recent developments, and major issues," *Simulation, 49*(3):109–116 (September 1987).
3. Biles, William E. "Introduction to Simulation," *Proceedings of the 1987 Winter Simulation Conference* (December 1987).
4. Blanchard, Chuck and others. "Reality Built For Two: A Virtual Reality Tool," *Computers Graphics, 24*(2):35–38 (March 1990).
5. Bolt, et al. "A History of the ARPANET: The First Decade," (April 1983).
6. Bratley, Paul, et al. *A Guide to Simulation.* New York: Springer-Verlag, 1983.
7. CIS Graphik und Bildverarbeitung GmbH. *Dimension 6 User's Manual.* Technical Report. Viersen West Germany, December 1987.
8. Crooke, J. G. and B. Valentine. "Simulation in Micro-Computers," *Journal of the Operations Research Society, 33*(4):855–858 (1982).
9. Duisberg, Robert A. "Animation Using Temporal Constraints: An Overview of the Animus System," *Human Computer Interaction, 3*:275–307 (1987-1988).
10. Filer, Capt Robert E. *Virtual Environment Display System.* MS thesis, AFIT/GE/ENG/89D-2, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.
11. Fiume, Eugene. "Active objects in the construction of graphical user interfaces," *Computers and Graphics, 13*(3):321–327 (1989).
12. Hurion, R. D. *The design, use and required facilities of an interactive visual computer simulation language to explore the production planning problem.* PhD dissertation, University of London, England, December 1976.
13. Law, Averill M. and W. D. Kelton. *Simulated Modeling.* New York: McGraw Hill, 1982.
14. Leffler, Samuel J. *The Design and Implementation of 4.3BSD Unix Operating System.* Reading MA: Addison-Wesley, 1989.
15. Newman, William M. and Robert F. Sproull. *Principles of Interactive Computer Graphics* (Second Edition). New York: McGraw-Hill, 1979.

16. O'Keefe, Robert M. "What is Visual Interactive Simulation," *Proceedings of the 1987 Winter Simulation Conference*, pages 461–464 (December 1987).

17. Silicon Graphics, Inc. *Graphics Library Reference Manual: C Edition.* Version 3.0. Document No. 007-1203-030.

18. Silicon Graphics Inc. *Silicon Graphics Periodic Table of the IRISes.* Product Announcement. Mountain View CA, July 1990.

19. Sutherland, Ivan E. "The Ultimate Display." In *Proceedings of the IFIP Congress 65*, pages 506–508, 1965.

20. VPL Research, Inc. *Dataglove Model 2.* Product Announcement. Redwood City CA, March 1989.

21. Wang, Chu P., et al. "Design for Interactive Performance in a Virtual Laboratory," *Computers Graphics, 24*(2):39–40 (March 1990).

22. Wardin, Capt Charles R. *Battle Management Visualization System.* MS thesis, AFIT/GE/ENG/89D-56, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.

## *Vita*

Captain William J. DeRouchey [redacted] He graduated from Miller High School in 1976. Following high school, he enlisted in the Air Force and was assigned to 44th Transportation Squadron at Loring AFB. He was reassigned to 42nd Missile Wing at Ellsworth AFB in November, 1979. He was selected for an Airman's Scholarship Commissioning Program in August, 1982. After receiving his commission, he was assigned to the Logistics Support Branch of the Human Factors and Logistics Division of the Air Force Human Resources Laboratory. He worked there until he entered AFIT as a full-time student in May 1989.

[redacted]

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | December 1990 | Master's Thesis |

**4. TITLE AND SUBTITLE**
A Remote Visual Interface Tool for Simulation Control and Display

**5. FUNDING NUMBERS**

**6. AUTHOR(S)**
William J. DeRouchey, Captain, USAF

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Air Force Institute of Technology, WPAFB OH 45433-6583

**8. PERFORMING ORGANIZATION REPORT NUMBER**
AFIT/GCS/ENG/90D-3

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**
Approved for public release; distribution unlimited

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

Visual Interactive Simulation (VIS) is an animation method used to create a dynamic display of a system model, and allow the viewer to control and interact with the running simulation. This research discusses the design and development of the Visual Interactive Simulation Interface Tool (VISIT). VISIT was designed to provide the graphical representation and user interface for a simulation that is running on another processor utilizing internet communications. The system provides support for various input devices to control the simulation display and environment. Viewer interaction to the s mulation is provided by a collection of commands that allow the viewer to initialize, start, stop, abort, and restart the simulation. The viewer also has the ability to establish checkpoints. Upon reaching a checkpoint the viewer can step through the output display and/or manipulate the objects within the simulation. Since the simulation processor may be any system capable of implementing the proper network protocol, specialized processors with no graphics display devices can benefit from this architecture. While primarily a system to provide visual representation of a simulation in real-time, VISIT also provides simulation developers a medium to verify and validate system models.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| Visual Interfaces, Discrete Simulations, Animation, Interactive Programs | 84 |
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UL |